Pliny Design Documentation

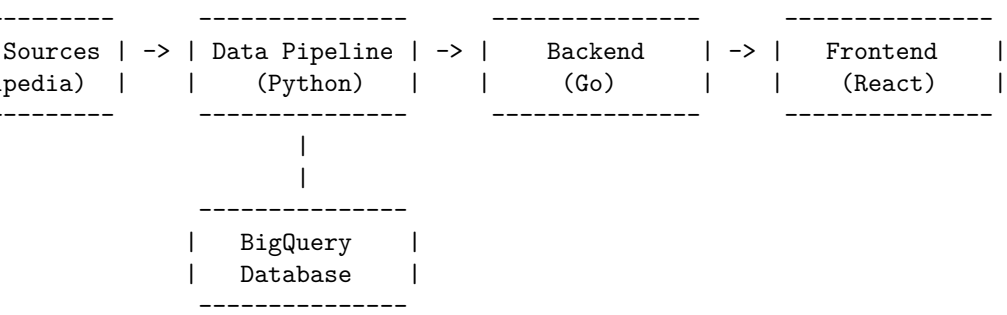
1. System Architecture

1.1 Overview

Pliny is architected as a multi-tier application comprising four main components: Data Sources, Data Pipeline, Backend Service, and Frontend Client. This architecture enables separation of concerns, independent scaling, and modular development.

1.2 Component Diagram

The system follows a pipeline architecture where data flows from external sources through processing stages to user presentation on the frontend.



1.3 Data Flow

1. Raw metadata is extracted from Wikipedia APIs and data dumps
2. The data pipeline extracts this metadata from the data sources and stores it in BigQuery
3. The data pipeline then performs transformations on this raw data to compute trends, which are also stored in BigQuery
4. The backend queries processed data from BigQuery
5. The frontend requests trend data from the backend
6. The frontend visualizes the trend data for end users

2. Component Design

2.1 Data Sources

Wikipedia provides multiple sources of metadata that Pliny consumes:

2.1.1 API Sources

- **Analytics API**: Provides aggregated information such as page views
- **MediaWiki API**: Provides lower-level information such as per-page individual commits

2.1.2 Data Dumps

- **Pageview Complete**: Daily dumps of all pageview data
- **Daily Edit History**: Complete edit histories for all articles on a particular day
- **Monthly Edit History**: Event log for all edit activity in a month, can be replayed to construct per-day information.

2.2 Data Pipeline (Python)

2.2.1 Responsibility The data pipeline is responsible for extracting data from Wikipedia sources, transforming it into a standardized format, identifying trends, and loading the processed data into BigQuery.

2.2.2 Key Components

- **Source Connectors**: Modules for each data source with specific handling logic
- **Data Processors**: Transformation logic for each data type
- **Trend Analysis**: Algorithms to identify significant patterns
- **BigQuery Writer**: Component to upload processed data to BigQuery

2.2.4 Intermediate Table Database Schema

```
- intermediate_table
  - date: DATE
  - page_name: STRING
  - view_count: INTEGER
  - edit_count: INTEGER
  - revert_count: INTEGER
  - editor_count: INTEGER
  - net_bytes_changed: INTEGER
  - abs_bytes_changed: INTEGER
  - abs_bytes_reverted: INTEGER
```

Note that this table is partitioned on `date` to significantly improve the performance and cost effectiveness of the trends analysis queries. This is because a trends query looks back at most 7 days, needing to only query a very small part of the table (opposed to having to do a full table scan).

2.2.5 Final Tables Database Schema There are a variety of final tables derived from the intermediate table, one for each trend. They all vary in schema based on the uses, but each has at least a `date` and `page_name` schema.

Note that all the final tables are also partitioned on `date` to improve cost effectiveness.

2.2.6 Additional Info Much more in-depth info can be found in the `README` for the data pipeline under `data-pipeline/README.md`

2.3 Backend Service (Go)

2.3.1 Responsibility The backend provides a REST API for accessing processed trend data.

2.3.2 API Design

- **GET /{trend}/{date}/{limit}**
  - All trend queries use this format.
  - Query parameters:
    - `trend`: Examples include: `topViews`, `topEdits`, `topVandalism`, etc
    - `date`: Date to query for. This is the "end" date for each trend (since most trends look back 3-7 days).
    - `limit`: Maximum number of pages to return
  - Response format: JSON array of trend info for each top page.
- **GET /availableDates**
  - Response format: JSON array of available dates (string format)

2.3.3 Key Components

- **Router**: Handles HTTP routing (using Go Gin framework)
- **Controllers**: Processes requests and constructs responses
- **Services**: Business logic for retrieving and formatting data
- **Database Client**: BigQuery connector for querying trend data

2.4 Frontend Client (TypeScript/React)

2.4.1 Responsibility The frontend provides an intuitive interface for users to explore Wikipedia metadata trends through various visualizations.

2.4.2 Component Structure

- **App**: Main container component
- **DateSelector**: UI for selecting time period
- **Visualizations**: Container for all visualization components

2.4.3 State Management

- React state hooks for UI state
- Fetch API for data retrieval from backend

2.4.4 Visualization Libraries

- Recharts for standard chart components, customized with CSS to provide unique Pliny look.

3. Data Model

3.1 Data Entities

3.1.1 Raw Metadata

- **PageView**: Record of a page being viewed
  - page_title: String
  - date: DateTime
  - count: Integer
- **PageEdit**: Record of a page being edited
  - page_title: String
  - timestamp: DateTime
  - user: String
  - bytes_changed: Integer
  - revert: Boolean

3.1.2 Processed Data

- **IntermediateTableRow**: Identified pattern in metadata
  - page_name: String
  - view_count: Int
  - edit_count: Int
  - editor_count: Int
  - revert_count: Int
  - net_bytes_changed: Int
  - abs_bytes_changed: Int
  - abs_bytes_reverted: Int

4. Performance Considerations

4.1 Data Volume

- Wikipedia generates approximately 300MB of metadata daily
- Processing focuses on English Wikipedia initially
- Data retention planned for 2+ years of historical data

4.2 Optimization Strategies

- Batch processing of historical data
- Incremental processing of new data
- Caching of commonly requested trend data
- Efficient database schema design to limit scanned data.

5. Security Considerations

5.1 Data Security

- All data used is public Wikipedia metadata
- No personally identifiable information is collected or stored
- BigQuery access requires appropriate credentials

5.2 API Security

- Public read-only access to trend data
- Rate limiting to prevent abuse
- No write operations exposed via API

6. Deployment Architecture

6.1 Infrastructure

- GCP Virtual Machine for hosting backend and frontend
- BigQuery for data storage
- NGINX as web server and reverse proxy

6.2 Deployment Process

- Backend: Go deployed with standard `go run` command.
- Frontend: Static files served via NGINX
- Data Pipeline: Scheduled execution via cron job on machine.

7. Future Enhancements

7.1 Technical Enhancements

- Support for non-English Wikis
- Extended historical data coverage
- Advanced trend detection algorithms
- Real-time trend updates (currently there is a lag of ~2 days)

7.2 Feature Enhancements

- User accounts for saved preferences
- Trend comparison tools
- Custom visualization creation
- Data export functionality